New Cardinality Notations and Styles for Modeling NoSQL Document-store Databases

Abdullahi Abubakar Imam[1,2], Shuib Basri[1], Rohiza Ahmad[1], Norshakirah Aziz[1], María Teresa González-Aparicio[3]
[1]CIS Department, Universiti Teknologi PETRONAS, Bandar Seri Iskandar, 31570, Perak, Malaysia.
[2]CS Department, Ahmadu Bello University, Zaria-Nigeria.
[3]Computing Department, University of Oviedo Gijon, Spain.
[1]aiabubakar3@gmail.com; {abdullahi_g03618, shuib_basri, rohiza_ahmad, norshakirah.aziz}@utp.edu.my; maytega@uniovi.es

***Abstract*—Nowadays, data with several characteristics such as volume, variety etc. are generated daily, i.e. big data; its complexity cannot be overemphasized. On the other hand, schema free NoSQL databases keep emerging at almost the same phase to accommodate such data which cannot be efficiently managed by relational databases. However, this advancement brings about the challenge to model such flexible databases and capably manage big data despite its complexity. In doing that, developers tend to apply their relational modeling skills; nonetheless, such skills may not be directly compatible with NoSQL databases due to their schema flexibility, linear scalability among others. To alleviate this difficulty, we propose a standard for modeling NoSQL databases, document-stores in particular. The standard can be classified as i) cardinality notations, and ii) relationship modeling styles. With such standard, NoSQL document-store databases can be properly designed, automated database testing can be applied, and database performance and stability can be considerably improved. To achieve this, experimental method is applied. Also, exploratory approach was used to explore the available literature as well as experts consultations. All possible entity relationships were extracted, aggregated and compiled from a heuristic evaluation of existing 4 different document-store databases. An experiment was conducted to assess the effect of the proposed standards, results indicate a profound improvements in various aspect of document modeling when the proposed standards are adopted, especially in a large scaled databases.**

Keywords— Cardinality Notations; Modeling Styles; NoSQL Databases; Big Data; Document-Store; Modeling guidelines.

I. INTRODUCTION

According to ISO, "great things happen when the world agrees" [1]. Thus the time for NoSQL standards is now. NoSQL databases have become so popular for many reasons such as their capability to handle data with numerous characteristics like variety, velocity, volume and variability, i.e. big data [2], [3], [4]. However, their heterogeneity, flexibility coupled with developers limited NoSQL skills has led to low quality designs of the NoSQL database structure [3], [5], [6], [7]. Beforehand, programmers are acquainted with skills of developing SQL databases for decades where schemas are enforced by database engine, but with the emergence of schema free NoSQL databases, experts tend to apply their SQL skills in modeling NoSQL databases, especially with document-store (D-store) databases as they have commonalities with SQL databases [8], [9], [10], [11].

For instance, William (2014), Lead Technical Engineer at MongoDB states that when modeling relationships such as one-to-many in D-store, notation like 1:M and its concept is adopted from SQL, however, it may not always be the case in NoSQL databases. The M may be further classified into Few, Many or Squillions to compliment the beauty of NoSQL databases. In addition, embedding a child document into the parent document doesn't always signify best practice. At some point, referencing might be more suitable for better performance.

As a result of the aforementioned impediments among others, a research was conducted by [5] to mitigate the modeling issues associated with document-store databases using Formal Concept Analysis, however, in their approach, only existing relational database modeling techniques are considered which may not always work or need more in-depth classifications as explained in the previous paragraph. Consequently, a standardized guide for modeling relationships in document-store databases is proposed in this paper. This has become necessary as data increase exponentially in size and complexity every day; thus progressively complicate NoSQL database modeling and increase chances of erroneous designs, which may negatively influence system performance [11], thereby lead to system crush at worst.

In document-store databases, depending on the nature of the data, documents are modeled as a collection of related files [7][11]. There exist a number of document-store databases which include MongoDB, Apache Cassandra, Couchbase, CouchDB among others [12]. In this paper, we use MongoDB for implementation of our proposed cardinality standards. MongoDB is widely embraced for its flexibility, availability of supports and compatibility with many programming languages such as .Net, Java, JavaScript PHP, Python and so on [13]. It is remarkable that, ebay uses MongoDB for its online services like session management, shopping carts, preferences and product catalog. Also, Facebook, a social media website uses MongoDB for its major project called Facebook Parse (FP). With FP, programmers can build, manage and house their mobile apps for as long as they wish on FP. This technological support generates tons of data daily from multiple users.

978-1-5090-1134-6/17/$31.00 ©2017 IEEE

To achieve our goal, four top most document-store databases [13] are selected [4] and individually explored to identify commonalities as well as disparity points. This leads to the extraction of modeling harmonization areas; thereby, ground our theories to have basis which can guide the proposition of the new standards. Experimental approach was adopted where one software application with one document-store database was engineered to rigorously test the proposed standards. Cardinality notations and relationship styles proposed in this paper were modeled and implemented. It is at this point evident that, NoSQL databases, especially document-store databases, require standard modeling guide for better database design and appropriate relationship modeling.

The key contributions of this paper include:

- New cardinality notations for modeling document-store databases, taking into account embedding and referencing relationship styles.
- New relationship modeling styles.
- Trade-off analysis between modeling styles such as embedding, referencing and bucketing; thus help developers to choose between the styles while modeling their document-store databases.
- Evaluation of the proposed relationship standards using the widely used NoSQL document-store databases, MongoDB [13].

II. Related Work

NoSQL document-store databases are highly flexible [7]. They are based on a flexible model that allows schemas to be written and managed by the client side application developers [5][14]. However, this may lead to incorrect or inappropriate schema design especially when modeling relationships between datasets and entities [5][9].

Document database experts have shared their experiences on the internet about the most common questions asked by the client side application developers. Some of these questions are (i) how to model one-to-N relationship in document databases? Or (ii) how does one know when to reference instead of embedding a document? Or (iii) do document databases allow Entity Relationship modeling at all? In an attempt to address these and alike questions, experts highlighted the necessity to come together and standardize these powerful data stores [5][11][15][9]. This is partly because many of the questions keep reappearing repeatedly in multiple knowledge sharing platforms.

As such, few attempts were made to incorporate relational modeling techniques into the NoSQL databases. [5] proposed conceptual modeling using Formal Concept Analysis (FCA). This was proposed to assist developers model document based databases. It adopted three (3) types of relationships from relational databases which are (i) one-to-one → 1:1, (ii) one-to-many → 1:M, and (iii) many-to-many → M:M relationships. These relationships were directly inherited from relational database and applied onto document-store databases. This method reveals the effectiveness of the aforesaid relationships when applied to document-store databases, however, the type of data stored in document-stores are much more complex and bulkier than the one stored in relational databases; thus require more detailed cardinality breakdowns. Also, foreign keys are not directly supported in document-store databases. In addition, other contributing factors to document-store modeling such as embedding are not considered in this research despite its importance to NoSQL database modeling practice.

In a similar concept, some contributors, such as technical experts from JSON [11] and mongoDB [9] explained some ways to achieve relatively good data modeling relationships, however, the approaches are sort of proprietary, focusing on the functionalities of the database in which they set to promote. There is need to have a generalized approach which can be followed by at least one category of NoSQL databases [5][8][16].

On the contrary, [17] agrees that, data model relegates bulk of implementation to NoSQL programmers, therefore, aggregate data modeling style is proposed using Idef1X which is the standard data modeling language. Again in this study, relational database notations are used as opposed to the understanding that states, NoSQL databases have bigger and more complicated datasets which require more detailed aggregate modeling techniques [8]. Whereas, an interactive, schema-on-read approach was proposed in [2] for finding multidimensional structures in document stores. Besides, [18] proposed data migration architecture which will migrate data from SQL to NoSQL document-store databases while taking into account the data models of both the two categories of databases.

It is therefore concluded that, as we move towards standardization in almost every aspect of technology [19] [20][21], NoSQL databases should not be left behind due to the heterogeneity nature of their data model and complexity of the data which they are designed to handle. Standardizations such as relationship modeling and data access should be rather encouraged to have a common agreement for best practice in storing, managing and retrieving data from such powerful databases.

III. Proposed Standard

NoSQL databases, specifically document-store databases, provide high scalability, low latency, availability and partition tolerance [6]. Moreover, they support flexible schema where databases are modeled freely without following any standard guide [9][5]. As a result, developers tend to apply any available skills such as relational database modeling skills to model such flexible databases [11][9]. By doing this, some key features like speed of document-stores become affected as relational database modeling skills cannot be directly applied in modeling NoSQL document-store databases and attain maximum benefits of their potentials [11][22][9]. However, some commonalities can be harnessed while eliminating some individual peculiarities. This is to simplify development hassles and minimize erroneous implementations.

This research aims at standardizing cardinality notations and styles which will be used when modeling NoSQL document-store databases. The fundamental principles of one document with respect to another are a critical aspect of relationship modeling. Therefore, in this study, standards are proposed while taking into cognizant the existing modeling expertize such as one-to-one, one-to-many etc. Initially, the cardinalities are presented followed by relationship styles.

A. Cardinalities

In modeling NoSQL document-store databases, the following cardinalities are proposed as in Table I below.

TABLE I. NOSQL DOCUMENT-STORE CARDINALITIES

ID	Cardinalities	Notations	Examples
1)	One-to-One	1:1	Person ←→ Id card
2)	One-to-Few	1:F	Author ←→ Addresses
3)	One-to-Many	1:M	Post ←→ Comments
4)	One-to-Squillions	1:S	System ←→ Logs
5)	May-to-Many	M:M	Customers←→ Products
6)	Few-to-Few	F:F	Employees ←→ tasks
7)	Squillions-to-Squillions[a]	S:S	Bank Transactions ←→ Logs

[a.] Squillion may be million or billions of records

Table I outlines the proposed cardinalities for NoSQL document-store databases. To describe document-stores relationships, familiar symbols are used as notations since relational modeling expertize already exist. The proposed cardinalities are chronologically (using ID in Table I) explained bellow.

1) One-to-One (1:1): To begin with, one-to-one relationship is a familiar type of cardinality. It is used to describe a relationship between two tables in traditional databases. Correspondingly, identical terminology is nominated in this study to describe a relationship between two entities (documents). One-to-one relationship means Entity-A relates to only Entity-B and vice versa. The following diagram illustrates the relationship.

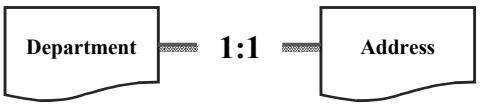

Fig. 1 – One-to-One Pattern

The one-to-one model illustrated in Fig. 1 can be further elaborated using the following schema example.

```
1 {
2   _id: "12345",
3   name: "Computer Science",
4   address: {
5     street: "road number 11",
6     city: "Seri Iskandar",
7     State: "Perak"
8   }
```

2) One-to-Few (1:F): The 1:F cardinality describes a relationship where one side of the model can contain more than one entity while the opposite can only contain one entity. The relationship is modeled as follows.

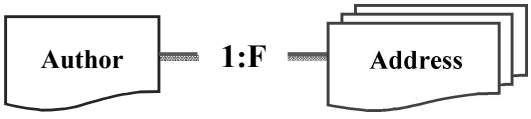

Fig. 2 – One-to-Few Relationship Pattern

Such relationship might be peculiar to NoSQL document-store databases as it might have no effect in the relational databases. Author and addresses are a perfect example that describes one-to-few. In document-stores, documents should be merged based on any of the relationship styles as explained in Section III Subsection B (Relationship Styles). For instance, one-to-few may be described as follows using the document-stores schema.

```
1 {
2   _id: "12345",
3   name: "Kazimbaya Fardjallah",
4   age: 35,
5   address: [
6     {street:"Rd 11",city:"Trono",State:"Prk", Cry:"MY"},
7     {street:"Rd 28",city:"Johor",State:"Joh", Cry:"MY"},
8     {street:"Rd 36", city:"Zaria", State:"KD", Cry:"NG"}
9   ]
10}
```

3) One-to-Many (1:M): One-to-many relationship has a similar concept with 1:F relationship as explained in the previous section. However, in one-to-many relationship, the "many" (M) part contains more documents (like hundreds or thousands). For example, consider a blog post where visitors respond to the post with comments, such comments will carry along the details of the commenters which may reach up to a couple of thousands but never too far. This type of relationship is modeled as follows.

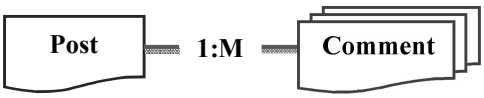

Fig. 3 – One-to-Many Relationship Pattern

1:M relationship can be further explained using the following schema example. The schema represents a post with a number of comments attached.

```
1 {
2   _id: 12345,
3   Title: "Introducing New Cardinalities for D-Stores",
4   Date: ISODate("2017-03-01T10:01:22Z"),
5   Posted_by: "A. A. Imam",
6   Comments: [
7          {name:"David Luise",
8            comment:"This is an important topic",
9            email:"david@gmail.com",
10             date: ISODate("2017-03-01T10:01:22Z")},
11         {name:"Abu zarrin",
12           comment:"The topic suits current situation",
13           email:"aduzz@gmail.com",
14             date: ISODate("2017-04-01T10:01:22Z")},
15          .....//hundreds or thousands of posts
16     ]
17  }
```

4) One-to-Squillion (1:S): Although there is no limit in the number of documents that can be created in one collection, a single document could be overflown when the document size limit is attained such as 16MB as the case of MongoDB. For example, consider a decentralized system that logs the activities of all its users and send timely reports to a central server. Such system can generate millions of records within a very short time. *1:S* can be modeled as follows.

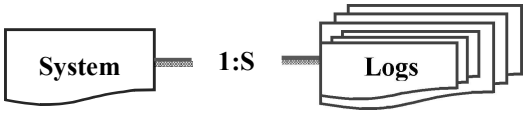

Fig. 4 – One-to-Squillions Relationship Pattern

The following schema is presented to exemplify one-to-squillion relationship.

```
{
  _id: ObjectID("12345"),
  activity_id: "Act005",
  name: "Cash Deposit",
  date:ISODate("17-02-04")
}
```

```
{
  _id: ObjectID("12346"),
  activity_id: "Act006",
  name: "Cash Withdrawal",
  date:ISODate("2017-02-04")
}
```

```
{
  name: "Bank Transaction Management",
  catalog_id: "9437",
  activities: [
        ObjectID("12345"),
        ObjectID("12346"),
        .. .. .. //squillions of logs reference ID
    ]
}
```

5) Many-to-Many (N:M): In many-to-many relationship, two sided connection between two entities is embraced. Many-to-many relationship is achieved by linking the references of the "one" side to the "many" side and the vice versa.

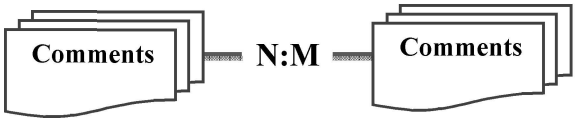

Fig. 5 – Many-to-Many Relationship Pattern

For instance, let us consider assignments-tracking system where there is a staff-collection with number of staff and assignments-collection which holds all assignments. Now, any or multiple staff can be assigned one or more assignments, such scenario can be represented as follows where assignment reference IDs are added to staff entity.

```
{
  _id: ObjectID("AS01"),
  name: "Design Curriculum",
  date:ISODate("217-05-05"),
  Owners: [
     ObjectID("1235"),
     ObjectID("1236")
   ]
}
```

```
{
  _id: ObjectID("AS02"),
  name: "Teach S/W Eng",
  Owners:
ObjectID("1235"),
  date:ISODate("2017-06-06")
}
```

```
{
  _id: ObjectID("1235"),
  name: "Shuib B",
  assignments: [
     ObjectID("AS01"),
     ObjectID("AS02")
     ... //… and so on
   ]
}
```

```
{
  _id: ObjectID("1236"),
  name: "Rohiza Ahmad",
  assignments: [
     ObjectID("AS01")
     ... //… and so on
   ]
}
```

Oppositely, to answer some questions like "what are the assignments handled by more than one person?" staff IDs are also added to assignments which indicate that the connection between the two entities is bidirectional.

By referring to the Table I, its denoted that *F:F and S:S* have similar structure with *N:M.* However, data access patterns and the nature of applications data may significantly contribute in choosing the most appropriate model when the entities on both sides are more than one.

B. Relationship Styles

Data access patterns and the nature of application's data are considered the major indicators of whether or not document-store schema should be modeled together, separate or bucketed. In this study, these styles of relationship are termed as shown in Table II and are briefly explained one after the other thereafter.

TABLE II. RELATIONSHIP STYLES

ID	Styles	Notations	Examples
1)	Embedding	EMB	Author ⟷ Addresses
2)	Referencing	REF	Post ⟷ Comments
3)	Bucketting	BUK	System ⟷ Logs

Each of the terminology presented in Table II above is briefly explained as follows, starting from the first in the list (Embedding):

1) Embedding (EMB): Embedding can be defined as a process of including a sub document or multiple sub documents inside another document. The document that is embedded is referred to as "child" document, while "parent" term is used to refer to the document that incorporates other sub documents. Two types of embedding such as one-way and two-way embedding are observed. The pattern which describes both styles is presented in Fig. 6 below and also explained afterwards.

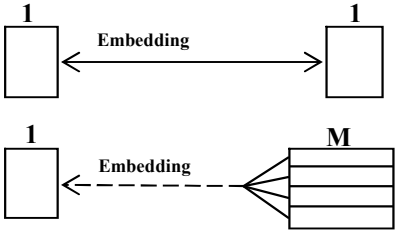

Fig. 6 – Embedding Style Pattern

For example, a department in a university may incorporate other sub documents that hold the details of all the programs/courses available. Such type of entity attachment may be referred to as one-way embedding style of relationship. Whereas in two-way embedding, books and authors can be considered where one author appears in many books and many books appear in the author's entity.

2) Referencing (REF): Unlike embedding which includes sub-documents into the parent document, referencing connects two or more separate documents together using a unique identifier. For instance, when a document-A is said to reference document-B, the ID of document-A will be present in document-B or/and vice versa, depending on the system developer. Referencing style of relationship can be described using the following pattern.

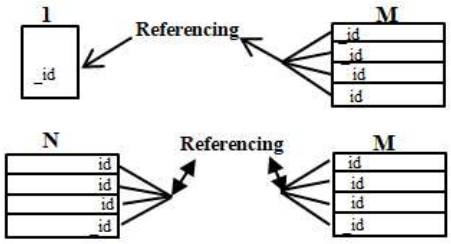

Fig. 7 – Referencing Style Pattern

To explain it further, *referencing* is classified into two, namely child-referencing and parent-referencing. Both the two types of referencing are combined and explained using a university system, where tasks are assigned to staff and/or vice versa.

3) Bucketting (BUK): Bucketing refers to splitting of documents into smaller manageable sizes either by quantity, days, hours etc. It balances between the rigidity of embedding and flexibility of referencing. Bucketing helps in document retrieval and saving. For example, using *1:S* relationship, the *S* side can be bucketed into smaller quantity such as 5000s to improve document retrieval speed and portability in data presentations as the case of pagination. The following section explains the experimental set-up, tools and their specifications.

C. Experiments

A real-world example was implemented in order to look into how the proposed cardinality notations and relationship styles affect document-store relationship modeling. The experiment was conducted with the following hardware/software: an intel dual processor, core i7-3632QM; CPU running at 2.20GHz * 2; and 8GB of RAM. 64 bit of windows 10 was used as the operating system; Dev-C++ as IDE (Integrated Development Environment); C++ as the programming language; and NoSQL document-store database (mongoDB) as the database management system. Results of these experiments are presented in the next section.

IV. Results and Discussions

In this section, the results generated from the experiment are presented and discussed. The results are classified based on the cardinality notations presented in Table I and the styles discussed in Section III B. Each of the cardinalities is assessed to see which of the relationship style suits it most. It is observed that some cardinalities have a very similar relationship pattern. For example, *F:F* and *N:M* presented in Table I signify two sided many-to-many relationship with different *N* or *M* sizes. As such, in the interest of generalization, we distinctly experiment the cardinalities and put forward the results as follows. Each read/write operation is measured by time, microseconds (μs) in particular.

TABLE III. Results of One-to-One & One-to-Few (1:1 & 1:F) Relationships

Number of Documents	Relationship Styles					
	Embedding		*Referencing*		*Bucketing*	
	W-T:µs	*R-T:µs*	*W-T:µs*	*R-T:µs*	*W-T:µs*	*R-T:µs*
1:1 documents	742000	734000	813000	941000	-	-
1:F documents (7)	795000	771000	894000	976000	1371000	1271000

It can be seen from Table III above that, embedding document in one-to-one or one-to-few relationships is more viable than *referencing* or *bucketing* the related documents. Using *embedding*, write/read operations were achieved in lesser time than *referencing* or *bucketing*. This is because the numbers of entities are very few and *referencing* or *bucketing* them may incur more time in both *writing (W)* and *reading (R)* data. This is in line with the proposed styles, to *embed* a document when the "many" side of the relationship is few.

TABLE IV. Results of One-to-Many (1:M) Relationship

Number of Documents	Relationship Styles					
	Embedding		*Referencing*		*Bucketing*	
	W-T:µs	*R-T:µs*	*W-T:µs*	*R-T:µs*	*W-T:µs*	*R-T:µs*
1:M documents (5000)	1374000	1326000	1131000	1221000	1635000	1573000

In *1:M* relationship, a document with 5000 other related documents were considered. Unlike in the previous experiments where *embedding* dominated other modeling styles, this time, results indicate a profound improvement in the *referencing* technique as it leaves *embedding* and *bucketing* behind who scored 1374000W, 1326000R, 1635000W and 1573000R respectively. The reason is that, when such numbers (5000) of documents are *embedded*, the document becomes larger and larger which must be accessed each time read/write data is needed.

TABLE V. Results of One-to-Squillion (1:S) Relationship

Number of Documents	Relationship Styles					
	Embedding		*Referencing*		*Bucketing*	
	W-T:µs	*R-T:µs*	*W-T:µs*	*R-T:µs*	*W-T:µs*	*R-T:µs*
1:S docs (500000)	3601000	3579000	1871000	1931000	2412000	2161000

In *1:S* relationship, the difference between the relationship styles become clear where *embedding* looks not to be a viable option for modeling *1:S*. whereas, *bucketing* shows a slight improvement from the previous experiment, this indicates that as data size increase relevance of *bucketing* become more pronounce. However, *referencing* style has shown its powers when modeling *1:S* relationship. This is as a result of decentralization method embraced by *referencing* style since the increment of data does not affect the main document. So, it can be concluded that, *referencing* style is the choice for *1:S*.

TABLE VI. Results of Few-to-Few, Many-to-Many and Squillion-to-Squillion Relationship

Number of Documents	Relationship Styles					
	Embedding		*Referencing*		*Bucketing*	
	W-T:µs	*R-T:µs*	*W-T:µs*	*R-T:µs*	*W-T:µs*	*R-T:µs*
N:M docs (100000)	2710000	2504000	1700000	1831000	1950000	1773000

On the other hand, in *N:M* relationship, *bucketing* and *referencing* styles perform very well, mostly when retrieving documents. Whereas, *embedding* seems not go with such type of data size. This is because many of the documents are large in size, and *bucketing* partitioned them for faster retrieval, however, the partitioning did not work well when writing data. With all this competition, again, *referencing* style has shown better performance for both read and write events. It is therefore concluded that, *referencing* is a better option for *N;M* relationship, then followed by *bucketing*.

V. Conclusion and Future Work

It's brought to our notice that NoSQL databases, especially document-store databases opened a new problem area for data modelers. These days data with several characteristics is generated daily, its complexity cannot be overemphasized. Also, document-store databases keep emerging periodically. However, developers must model one or more document-stores based on such complicated data to maximize efficiency and minimize the chances of system breakdown and so on. As such,

this paper proposed new cardinality notations and relationship styles for modeling NoSQL document-store databases. To achieve this feat, experimental approach (exploratory and confirmatory) was applied in this research. This involves exploration of the available literature, heuristic evaluation of existing document-store databases as well as consultations of the document-store experts. Rigorous experiment was conducted to assess the proposed ideas.

Results indicate a significant improvement in the general performance of the NoSQL document-store databases when the standards are adopted, specifically when read/write events are performed. In addition, it is concluded that, the proposed cardinalities and modeling styles concepts can, without doubt, ease development process, minimize erroneous schema implementation and improve system performance, especially in a large scale applications. Our future focus would be to propose an easier way to model NoSQL databases. Undoubtedly, modeling NoSQL databases will continue to be the focus of future research.

ACKNOWLEDGEMENT

The authors wish to acknowledge the support from Universiti Teknologi PETRONAS (UTP) for funding this research through Yayasan and Graduate Assistantship Scheme (UTP-GA).

REFERENCES

[1] ISO, *International Organization for Standardization Strategy 2016 - 2020*. Switzerland: International Organization for Standardization, 2016.

[2] M. L. Chouder, S. Rizzi, and R. Chalal, "Enabling Self-Service BI on Document Stores," *Work. Proceed- c ings Jt. Conf. Venice, Italy*, 2017.

[3] P. Atzeni, F. Bugiotti, and L. Rossi, "Uniform access to NoSQL systems," *Inf. Syst.*, vol. 43, pp. 117–133, 2014.

[4] J. G. Enríquez, F. J. Domínguez-Mayo, M. J. Escalona, M. Ross, and G. Staples, "Entity Reconciliation in Big Data Sources: a Systematic Mapping Study," *Expert Syst. Appl.*, vol. 80, pp. 14–27, 2017.

[5] V. Varga, K. T. Jánosi, and B. Kálmán, "Conceptual Design of Document NoSQL Database with Formal Concept Analysis," *Acta Polytech. Hungarica*, vol. 13, no. 2, pp. 229–248, 2016.

[6] M. T. Gonzalez-Aparicio, M. Younas, J. Tuya, and R. Casado, "A New Model for Testing CRUD Operations in a NoSQL Database," in *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*, 2016, vol. 6, pp. 79–86.

[7] J. Han, E. Haihong, G. Le, and J. Du, "Survey on NoSQL database," *Proc. - 2011 6th Int. Conf. Pervasive Comp. Appl.*, pp. 363–366, 2011.

[8] P. Atzeni, "Data Modelling in the NoSQL world : A contradiction ?," no. June, pp. 23–24, 2016.

[9] Z. William, "6 Rules of Thumb for MongoDB Schema Design," *MongoDB*, 2014. [Online]. Available: https://www.mongodb.com/blog/post/6-rules-of-thumb-for-mongodb-schema-design-part-1. [Accessed: 23-Jan-2017].

[10] R. April, "NoSQL Technologies: Embrace NoSQL as a relational Guy – Column Family Store," *DBCouncil*, 2016. [Online]. Available: https://dbcouncil.net/category/nosql-technologies/. [Acc: 21-Apr-2017].

[11] R. CrawCuor and D. Makogon, *Modeling Data in Document Databases*. United States: Developer Experience & Document DB, 2016.

[12] N. Jatana, S. Puri, and M. Ahuja, "A Survey and Comparison of Relational and Non-Relational Database," *Int. J*,vl.1, no.6, pp.1–5, 2012.

[13] M. Gelbmann, "DB-Engines Ranking of Document Stores," *DB-Engines*, 2017. [Online]. Available: https://db-engines.com/en/ranking/document+store. [Accessed: 21-Feb-2017].

[14] T. A. Alhaj, M. M. Taha, and F. M. Alim, "Synchronization Wireless Algorithm Based on Message Digest (SWAMD) For Mobile Device Database," *2013 Int. Conf. Comput. Electr. Electron. Eng. Synchronization*, pp. 259–262, 2013.

[15] G. Matthias, "Knowledge Base of Relational and NoSQL Database Management Systems: DB-Engines Ranking per database model category," *DB-Engines*, 2017. [Online]. Available: https://db-engines.com/en/ranking_categories. [Accessed: 21-Apr-2017].

[16] M. J. Mior, "Automated schema design for NoSQL databases," *Proc. 2014 SIGMOD PhD Symp. - SIGMOD'14 PhD Symp.*, pp. 41–45, 2014.

[17] V. Jovanovic and S. Benson, "Aggregate Data Modeling Style," *SAIS 2013 Proc.*, pp. 70–75, 2013.

[18] M. Mughees, "DATA MIGRATION FROM STANDARD SQL TO NoSQL," 2013.

[19] J. Häkkilä, "Developing Design Guidelines for Context-Aware Mobile Applications," *Proc. 3rd Int. Conf. Mob. Technol. Appl. Syst. ACM, 2006*, pp. 1–7, 2006.

[20] J. Rodriguez, "Guidelines for designing usable world wide web pages," *Conf. Companion Hum. Factors Comput. Syst. ACM.*, pp277–278, 1996.

[21] J. Gong and P. Tarasewich, I. Science, "Guidelines for Handheld Mobile Device Interface Design," *Proc. DSI Annu. Meet.*, pp. 3751–3756, 2004.

[22] W. Naheman, "Review ofNoSQL Databases and Performance Testing on HBase," *Int. Conf. Mechat. Sci. E. Eng. Comp.*, pp.2304–2309, 2013.